National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES
THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL 339 (r. 86/01)

Canada

124

0-315-26857-3

National Library of Canada | Bibliothèque nationale du Canada

Canadian Theses Division | Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

De-Lei Lee

Date of Birth — Date de naissance

Dec. 16, 1952

Country of Birth — Lieu de naissance

China

Permanent Address — Résidence fixe

Institute of Computing Technology. Chines Academia Sinica
Beijing. China

Title of Thesis — Titre de la thèse

Fast Algrithms for Associative Memories

University — Université

The Universiey of Alberta

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

Master of Science

Year this degree conferred — Année d'obtention de ce grade

Spring, 1984

Name of Supervisor — Nom du directeur de thèse

Wayne Davis

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

April, 4, 1984

Signature

De-Lei Lee

NL-91 (4/77)

THE UNIVERSITY OF ALBERTA

FAST ALGORITHMS FOR ASSOCIATIVE MEMORIES

by

DE-LEI LEE

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

SPRING, 1984

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR DE-LEI LEE

TITLE OF THESIS FAST ALGORITHMS FOR ASSOCIATIVE MEMORIES

DEGREE FOR WHICH THESIS WAS PRESENTED MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED 1984

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(SIGNED) De-Lei Lee

PERMANENT ADDRESS:

Institute of Computing Technology
Chinese Academia Sinica
Beijing, China

DATED 9 Dec. 1983

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled FAST ALGORITHMS FOR ASSOCIATIVE MEMORIES submitted by DE-LEI LEE in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE.

Wayne H. Davis

Supervisor

W. R. Armstrong

R. F. Hayhurst

John Tartar

Jack Brzozowski

Date 9 Dec 83

Abstract

This thesis is concerned with the design and implementation of efficient algorithms for associative memories. A general model of associative memories of m n-bit words is assumed, and the time complexity of any algorithm under this model is measured in gate delay units.

A new threshold search algorithm with time complexity $O(\log n)$ is presented, as compared to $O(n)$, the recent result of Ramamoorthy et al. [31]. Based on this algorithm, a class of search algorithms with the same time complexity is developed. The extremum search algorithm by Frie and Goldberg [5] is modified and generalized so that the number of memory interrogations is reduced by 30% over the initial algorithm in the average case.

Another new algorithm is proposed for ordered retrieval, i.e., sorting. It retrieves k responders in order from the associative memory in time $O(n+k)$ which compares favorably to $O(k \cdot \log n)$, the best result by Lewin [7]. Based on the proposed ordered retrieval algorithm, a fast multiple response resolver is suggested which resolves k responders in time $O(k+ \log m)$. The suggested resolver is faster than the previous fastest resolver with time complexity $O(k \cdot \log m)$ by Anderson [26] in most cases. Cellular logic implementations of these algorithms are discussed, and an analysis of the underlying hardware complexity given.

Acknowledgements

I wish to express my appreciation to Dr. W. A. Davis for his advice, encouragement and many stimulating discussions throughout the course of this research.

Further thanks are due to the members of my examining committee, Drs. W. Armstrong, J. Tartar, T. Marsland, and especially Dr. J. Mowchenko from the Department of Electrical Engineering, for their helpful comments.

I must thank the late Dr. I-Ngo Chen, who was instrumental in my coming to this department.

I am greatly indebted, above all, to my parents, Lee Ruqi and Yen Yumei, to whom this thesis is dedicated, for instilling in me their belief in the value of education.

Finally, I am deeply grateful to my wife, Xiaoning, whose love and assistance made this work possible.

Table of Contents

Chapter		Page
1.	INTRODUCTION	1
2.	BACKGROUND	5
	2.1 The organization of associative memories	5
	2.2 Definition of searches	12
3.	FAST SEARCH ALGORITHMS	15
	3.1 Previous search algorithms	15
	3.2 A new threshold search algorithm	21
	3.2.1 The word-tree concept	22
	3.2.2 The algorithm	23
	3.2.3 Implementation of double-limit searches	31
	3.3 A new extremum search algorithm	32
	3.4 Conclusion	36
4.	EFFICIENT ORDERED RETRIEVAL ALGORITHMS	37
	4.1 Previous ordered retrieval algorithms	37
	4.2 A new ordered retrieval algorithm	40
	4.2.1 System overview	40
	4.2.2 The algorithm	46
	4.2.3 An analysis of the algorithm	51
	4.3 Conclusion	55
5.	FAST MULTIPLE RESPONSE RESULOTION ALGORITHMS	56
	5.1 Previous results	56
	5.2 Anderson's Multiple Response Resolver	59
	5.3 A new multiple response resolver	62
	5.4 Comparison with Anderson's resolver	63
6.	CONCLUSION	66
	BIBLIOGRAPHY	67

List of Figures

Figure		Page
1.	Associative memory model	6
2.	Ramamoorthy's cell	19
3.	Searching logic	26
4.	A simplified associative memory	40
5.	The j-th component of the BSVI	45
6.	Partition logic	49
7.	Anderson's resolver structure	60
8.	P-generator block	61

CHAPTER 1

INTRODUCTION

Currently, computers predominantly make use of two conceptually different types of memories, random access memory and associative memory. The most prevalent is random access memory which allows the word to be retrieved by address, i.e., physical location in memory. Conventional computers achieve their generality from this location-addressable capability of random access memories; however, this generality makes the operations of searching and sorting overly time-consuming. If the random access memory is being used to store a list of N unordered records, where each record contains a fixed number of fields, it will take $O(N)$ memory accesses to find a record with a specific value in a certain field. The operation can be reduced to $O(\log N)$ memory accesses by using binary search [16], provided the N records are sorted according to the field being searched. On the other hand, sorting N records stored in the random access memory is a much more costly operation for the theoretical lower bound for sorting is of $O(\log N!)$ memory accesses and comparisons [16]. Moreover, when all fields of the record are equally important with respect to a query, an inverted file system has to be employed in order to provide the means for binary search [25]. This, however, requires extra memory space and, increases system overhead.

The concept of associative memory or alternatively content-addressable memory, as originally introduced by Salde and McMahon [1] in 1956, lends a much better solution for the above problems. The distinguishing feature of such a memory is that it allows stored words to be retrieved by their contents, or part of their contents. The importance of associative memories lies not only in accessing data by content but also in doing so in parallel.

As a result, a searching operation can be done in one associative memory access. This content-addressable capability consequentally reduces somewhat the need for sorting, particularly in the case where sorting is used to provide the means for binary search. It also eliminates the requirement of extra memory space for index files because searching can now be done equally well for each record field. Accordingly, system overhead is also minimized [25].

Sorting is now required only for sorting output lists. However, some efficient techniques have been discovered that take only $O(N)$ accesses of the associative memory to output, in order, a stored list of N records.

Because of the content-addressable characteristic, and efficient parallel processing capability, associative memory has found its application in various fields, such as sorting, relational data searches, pattern recognition, machine translation, question-answering systems, job scheduling and many others [21],[23],[27].

The design and implementation of efficient fundamental algorithms for associative memories, the subject of this thesis, is the heart of different applications and as such, has been a focus of research in the area of associative processing from the time the field was established. Furthermore, VLSI technology with its promise for the near future in conjunction with new associative memory architectures make the economic realization of efficient hardware algorithms for associative memories worth investigating.

This thesis begins with a description of a general organization and some background in which associative memories operate, and proceeds, in Chapter 2, to define the fundamental searches in the context of an associative memory of m n-bit words.

A new scheme for constructing search algorithms for parallel associative memories is described in Chapter 3. The resulting equivalence searches, threshold searches and double-limit searches achieve the time bound of $O(\log n)$ compared to $O(n)$, the recent result of Ramamoorthy [31]. The extremum search algorithm by Frie and Goldberg [5] is modified and generalized. It is shown that the modified algorithm reduces the number of memory accesses by 30% in the average case.

Chapter 4 addresses the very important, classical problem imposed by retrieval of an ordered list from associative memories. After briefly reviewing previous

solutions to the problem, an efficient new ordered retrieval algorithm is proposed along with a cellular logic implementation. The algorithm is proved to be of time complexity $O(n+k)$ compared to $O(k \cdot \log n)$, the previous fastest method of Lewin[7], where k is the number of responders to be retrieved in order.

Chapter 5 deals with the problem of resolving multiple responses in associative memories. The fastest multiple response resolver by Anderson [26] is examined. A new resolver is suggested. An analysis of the efficiency of these two resolvers is presented. The proposed resolver is superior to its competitor in most cases.

CHAPTER 2

BACKGROUND

This chapter establishes the fundamentals of associative memories which will be used in subsequent chapters: the concept and typical functional components of general associative memories, the characteristics of two different types of associative memory architectures, and a formal definition of various associative memory searches.

2.1 The organization of associative memories

The retrieval of information from a memory requires that a particular word of the memory be identified as containing the required information. When identification is done, the selected word can be read out and/or processed as required. According to the way the identification is achieved, computer memories fall into two different categories.

In the first category, the conventional random access memory accomplishes this identification by a given specification of the desired word's physical location or address in memory. Associative memory, in the second category, accomplishes this identification by specifying partial information of the desired word.

This difference provides associative memory with a variety of powerful memory operations other than the *READ*

and *WRITE* provided by random access memory. This extensive class of operations, to be precisely defined

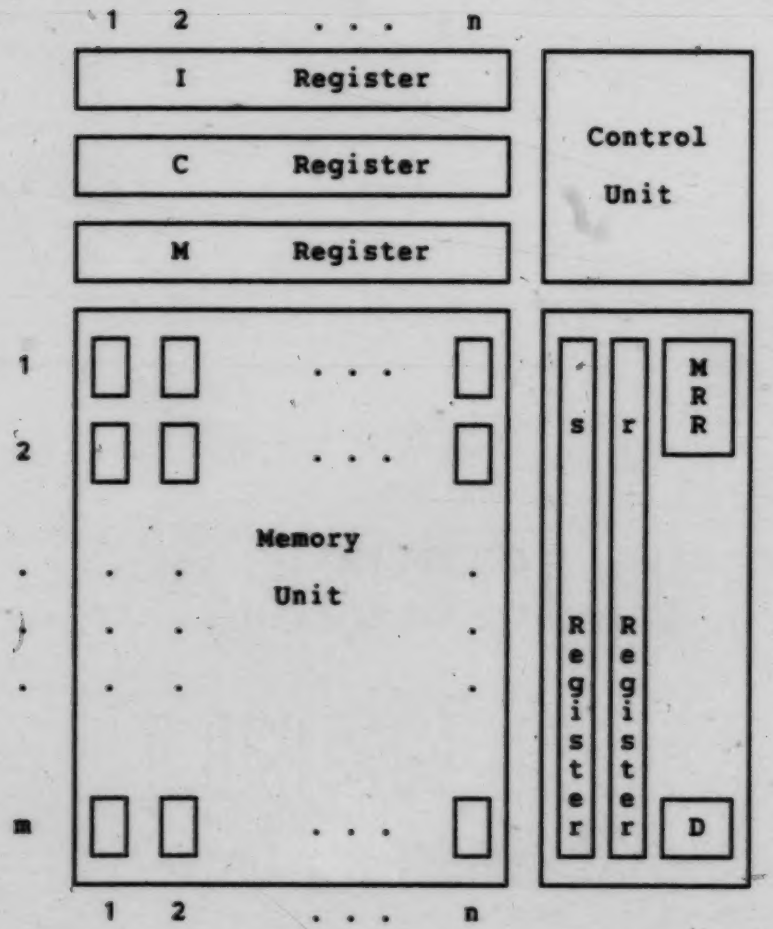


Fig. 1. Associative memory model.

in the next section, has made associative memory very attractive for a wide range of applications.

An abstract associative memory model is schematically shown in Fig. 1.

The memory unit is a two-dimensional, iterative configuration of m·n identical cells for storing m words of n bits each, denoted as B(i) for all $i \in \{1,\ldots,m\}$. The value of the j-th bit of B(i), B(i,j) is stored in the cell (i,j) at the i-th row and the j-th column, where j=1 is the most significant bit.

The memory unit is connected to five registers: I, C, M, s and r. Register I provides the means for buffering the information to be transfered between the host computing system and the associative memory. Registers C and M serve the purpose of specifying the contents or partial contents of a desired word. In particular, C holds an argument of n bits called the *search key word* against which B(i), $i \in \{1,\ldots,m\}$, will be compaired in parallel; M has the ability to mask some portions of the search key word in C. In other words, any bit of C can be masked as a *'don't care'* state as desired, and only those unmasked bits of C specify a search criteria. Let C(j) be the value of the j-th bit of C and M(j) the value of the j-th bit of M. M(j)=1 means C(j) is unmasked in searches, while M(j)=0 means C(j) is masked out as the 'don't care' condition. The 'don't care' state will be denoted throughout this thesis as ϕ.

Register s is used to select a set of words to be involved in searches. s(i) denotes the value of the i-th bit of s with s(i)=1 indicating that B(i) is in the set. Register r, on the other hand, is used to store a search result, where r(i) represents the value of the i-th bit of r. At the end of a search, r(i)=1 indicates that the corresponding word B(i) satisfies the search criterion.

In some associative operations, at the end of a search, a decision regarding the next action is based on the knowledge of the presence or absence of any *responders*, i.e., words responding to the search. The functional component, denoted as D, is included in Fig. 1 to provide this knowledge.

The component **MRR**, Multiple Response Resolver, deals with operations such as read and write. A search of the memory may yield more than one responder. The multiple response resolution arises when the associative memory outputs these responders one at a time. To select a responding word to read out, the **MRR** generates an m-bit vector of which there is exactly one element with a value 1 corresponding to the selected word. When this vector is generated, the selected word can then be read out without ambiguity. Evidently, the total time required to read out all responders depends greatly on how fast the vector can be generated to identify a word each time so that a read operation can take place. **MRR** techniques will be discussed further in Chaper 5.

Finally, the control unit coordinates the above functional components.

To support the content-addressable concept and the parallel processing capability, each word B(i) in the memory must have its own hardware comparison logic so that the contents of each B(i) can be compared with the value of the search key word specified by C and M simultaneously. It is now easy to see that associative memory belongs to the class of Single Instruction Stream-Multiple Data Stream (SIMD) machines, from the architecture viewpoint [27]. The comparison logic can be viewed as the processing elements of the SIMD machine, each of them either executes or ignores an instruction from the control unit on its own data, the associated word.

According to the way the comparison logic is provided to the words, which in turn determines the way searches are performed, associative memories can be further classified into the following two major categories:

1) Bit-parallel associative memories; and

2) Bit-serial associative memories.

In a bit-parallel associative memory, one bit of comparison logic is provided for each cell of the memory unit. In a search, the processing logic of cell (i,j) compares the contents of B(i,j) with one *interrogation* bit, the unmasked C(j). This is done simultaneously for every cell, and the final decision as to whether or not the word B(i) satisfies the given search criteria is made by all

cells in the i-th row. Assuming each cell in the memory unit in Fig. 1 is equipped with comparison logic, then it can be identified as a bit-parallel associative memory.

The bit-serial associative memory is based on the concept of parallel processing with vertical data introduced by Shooman [2]. Here, only one bit of comparison logic needs to be provided for each word B(i), and exists outside the memory unit. Assuming that none of the cells in the memory unit of Fig. 1 contains comparison logic, and one bit of hardware comparison logic is added together with s(i) and r(i) to form the processing element for each B(i) to gain the content-addressable capability, then the associative memory becomes a bit-serial one.

A bit-serial associative memory must be capable of reading all bits in any *bit-slice* into their corresponding processing elements in parallel. A bit-slice is made up of one bit of every word in the memory. The control unit is responsible for reading the unmasked bit-slices one at a time and broadcasting their corresponding portion of C one bit at a time into the processing elements in a synchronized manner. The storage element r(i) in processing element i is now used to remember the intermediate comparison state from one interrogating bit to the next in the course of a search.

The bit-parallel associative memory has the advantage of a simpler control structure and much faster search speed at the expense of the additional logic built into every cell. In contrast, a bit-serial associative memory has a

much slower search speed and a more sophisticated control structure, but considerably less hardware. The low search speed of the bit-serial associative memory is essentially imposed by the necessity of reading all unmasked bit-slices out of the memory unit and processing them outside the memory unit sequentially. It is, in fact, a compromise between bit-parallel associative memories and random access memories. The search time is measured in read cycles for the bit-serial associative memory and in gate delays for the bit-parallel one.

The most commonly used and widely discussed searches in the literature include the following:

1. equality search
2. inequality search
3. greater-than search
4. not-greater-than search
5. less-than search
6. not-less-than search
7. between-limits search
8. outside-limits search
9. maximum search
10. minimum search
11. nearest-above search
12. nearest-below search

There are other nonsearch operations that can be performed in associative memories. These include field addition [27], summation and product [29], counting [21], etc. Mechanisms that incorporate nonsearch operations are referred to as associative processors [28]. Nonsearch operations will not be further investigated in this thesis.

Before further discussing and analyzing some efficient associative memory algorithms, the searches summarized above will be more precisely defined in the next section.

2.2 Definition of searches

Definitions and notation for the searches to be examined fully in Chapter 3 will now be developed in the context of the associative memory model depicted in Fig. 1. Let S and Z be two sets defined as follows:

$S = \{B(i) | s(i)=1,\ 1 \le i \le m\}$, and $Z = \{j | M(j)=1,\ 1 \le j \le n\}$.

According to the magnitude of the unmasked bits of the search key word held in C, S can be partitioned into three disjoint subsets denoted by L, E, G respectively:

$$L = \{B(i) | \sum_{j \in Z} 2^{n-j}(B(i,j)-C(j)) < 0,\ B(i) \in S\},$$

$$E = \{B(i) | \sum_{j \in Z} 2^{n-j}(B(i,j)-C(j)) = 0,\ B(i) \in S\},$$

$$G = \{B(i) | \sum_{j \in Z} 2^{n-j}(B(i,j)-C(j)) > 0,\ B(i) \in S\}.$$

The searches can now be defined in terms of S, L, E and G as follows:

A. *Equivalence searches*:

1) equality search that produces E,

2) inequality search that produces S - E.

B. *Threshold searches*:

1) greater-than search that produces G,

2) not-less-than search that produces $G \cup E$,

3) less-than search that produces L,

4) not-greater-than search that produces $L \cup E$.

C. *Double-limit searches*:

For two given limits x and y, where x is less than y, let Lx, Ex, Gx and Ly, Ey, Gy be the sets defined as L, E and G with x and y respectively held in C. All double-limit searches can be defined in terms of these sets:

1) between-limit searches that produce

a) $Gx \cap Ly$,

b) $(Gx \cup Ex) \cap Ly$,

c) $Gx \cap (Ly \cup Ey)$,

d) $(Gx \cup Ex) \cap (Ly \cup Ey)$;

2) outside-limit searches that produce

e) $S - (Gx \cup Ex) \cap (Ly \cup Ey)$,

f) $S - Gx \cap (Ly \cup Ey)$,

g) $S - (Gx \cup Ex) \cap Ly$,

h) $S - Gx \cap Ly$.

D. *Extremum searches*:

1) maximum search that produces the set

$$\{B(i) \mid \sum_{j \in Z} 2^{n-j}(B(i,j)-B(k,j))>0, \forall k \neq i \text{ and } B(i),B(k) \in S\},$$

2) minimum search that produces the set

$$\{B(i) \mid \sum_{j \in Z} 2^{n-j}(B(i,j)-B(k,j))<0, \forall k \neq i \text{ and } B(i),B(k) \in S\}.$$

E. *Adjacency searches*:

1) nearest-above search that produces the set

$$\{B(i) \mid \sum_{j\in Z} 2^{n-j}(B(i,j)-B(k,j))<0,\ \forall k\neq i \text{ and } B(i),B(k)\in G\},$$

2) nearest-below search that produces the set

$$\{B(i) \mid \sum_{j\in Z} 2^{n-j}(B(i,j)-B(k,j))>0,\ \forall k\neq i \text{ and } B(i),B(k)\in L\}.$$

CHAPTER 3

FAST SEARCH ALGORITHMS

Prevous search algorithms are discussed in this chapter. A new algorithm for threshold search is proposed. The algorithm is of time complexity $O(\log n)$, as compared to the time complexity $O(n)$ of the previous fastest algorithm [31]. Based on this algorithm, extremum search is achieved by a sequence of 0.7n memory interrogations on the average, compared to exactly n interrogations conventionally [5]. Implementations of double-limit searches and adjacency searches are sketched, and an analysis of the underlying hardware complexity is also presented'.

3.1 Previous search algorithms

Search algorithms have been intensively studied in the past. Some important papers discussing general algorithms include Gauss [4], Frie and Goldberg [5], Falkoff [8], Estrin and Fuller [9], Wolinsky [13], Feng and Lee [17], Ramamoorthy [31], and others [24],[28].

Early bit-parallel associative memories had only equality search in hardware as the basic search, while the other searches were achieved by executing a corresponding sequence of the basic searches.

' Significant parts of this chapter are to appear in *IEEE Trans. Comput.* and *Proc. the 1st International Conf. on Computers and Applications*, 1984 [32],[33].

Perhaps the earliest proposal for extremum searches is due to Frie and Goldberg [5]. The maximum (or minimum) search was carried out by a sequence of exactly n basic searches. Also introduced in [5] was an interesting procedure for adjacency searches which needs, at best, one basic search and, at worst, 2n-1 basic searches. Algorithms for threshold and double-limit searches were also proposed and shown to be basic search sequences of length n and 2n respectively [8],[9].

The major advantage of such a scheme is the low hardware cost in building bit-parallel associative memories. As shown in [28], the basic cell requires only three gates for the comparison logic. It is a disadvantage, however, that search speed is uncomfortably slow for those composed searches, sixteen search operations out of eighteen.

A time versus space trade-off exists, to some extent, in the design of fast search algorithms for associative memories. Having a more powerful basic cell, in terms of comparison capability, the majority of search operations can be done in $O(n)$ gate delays instead of $O(n)$ basic searches. As a matter of fact, a hardware algorithm for a threshold search has been recently designed and shown to operate in n gate delays by Ramamoorthy [31]. All double-limit searches can then be done in 2n gate delays by performing the threshold search twice. Therefore, fourteen search operations out of eighteen can be achieved in only n to 2n gate delays. More interestingly, using the threshold search

as a basic operation provides the possibility to improve the speed of Frie and Goldberg's method for both extremum and adjacency searches. A new algorithm to be presented in Section 3.2 indeed achieves such an improvement. As reported in [31], the basic cell for the threshold search contains six gates for the comparison logic which doubles the amount of comparison logic of the cell in [28]. However, for a minor hardware investment, the gain is worthwhile.

The need for an extremely fast threshold search algorithm is quite obvious due to its tremendous impact on the overall performance of other composed searches. Before presenting a much faster new algorithm for threshold search, Ramamoorthy's algorithm is first examined.

The associative memory model in Fig. 1 and the related notation will be used to describe Ramamoorthy's algorithm. In addition, the symbol E(i,j) denotes the output generated by cell (i,j-1) which is being taken by cell (i,j) as input. The algorithm follows:

```
    Begin
1     C:=search key word;
2     M(j):=1, ∀j∈{1,...,n};
3     r(i):=0, ∀i∈{1,...,m};
4     E(i,1):=s(i), ∀i∈{1,...,m};
5     For j:=1 Until n Do
6       Begin
7         E(i,j+1):=E(i,j)·¬(M(i,j)·(B(i,j)⊕C(j)));
```

```
8        d(i,j):=E(i,j)·B(i,j)·M(j)·(B(i,j)⊕C(j));

                  j
9        G(i):=  U  d(i,k); (wired-or)
                 k=1
      end;
   end.
```

Upon termination of the algorithm, E(i,n+1) together with G(i) indicate the membership of word B(i) as follows:

1). $G(i)=1 \Rightarrow B(i) \in G$;

2). $E(i,n+1)=1 \Rightarrow B(i) \in E$;

3). $\neg E(i,n+1) \cdot \neg G(i)=1 \Rightarrow B(i) \in L$.

The r(i) can be properly set to store the result of a search as required.

The logical realization of the cell to implement steps 7, 8 and 9 of the algorithm is shown in Fig. 2. The delay from E(i,j) to E(i,j+1) is one gate delay.

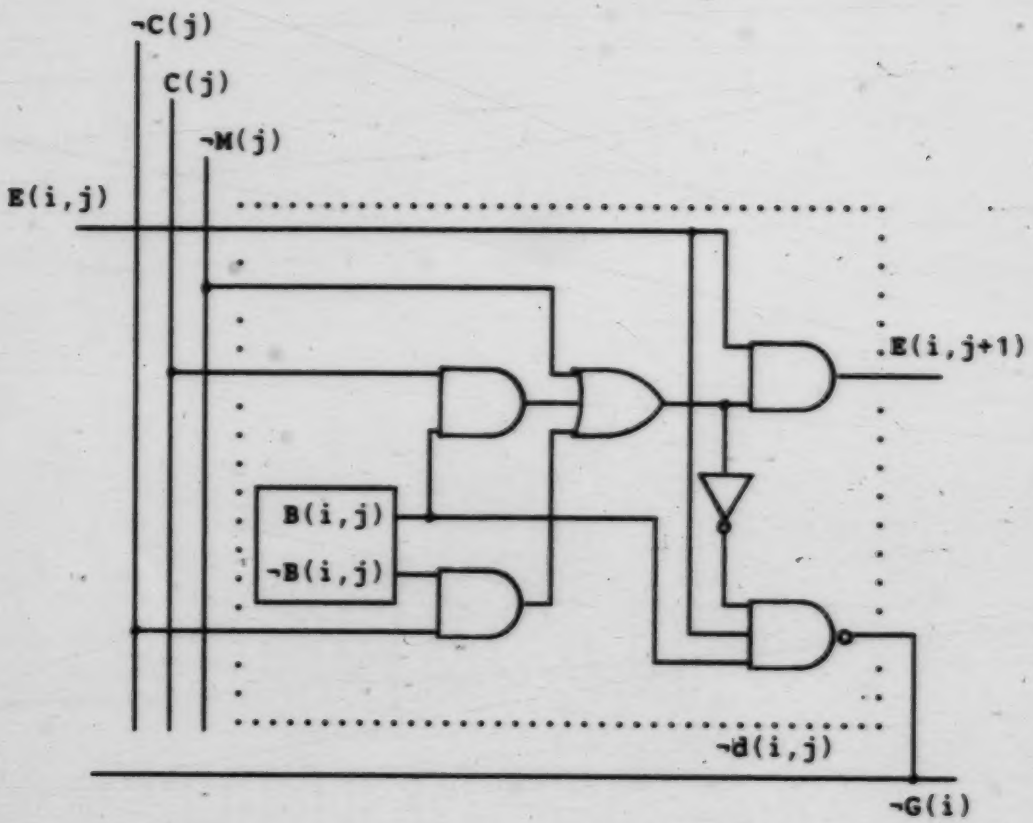


Fig. 2. Ramamoorthy's cell.

The following example shows a threshold search of four words, each seven bits long, where the symbol

$\begin{bmatrix} G \\ E \end{bmatrix}_{ij}$ represents the state of G(i) and E(i,j+1) enabled by cell (i,j) at the end of the j-th iteration of the algorithm.

Example: Threshold search operation

search key word: C = 1 0 1 1 0 1 1,

bit mask pattern: M = 1 1 1 1 0 1 1,

effective search key word: 1 0 1 1 ϕ 1 1,

words in the memory: B(1) = 1 0 1 1 1 1 1,

B(2) = 0 1 0 0 0 1 0,

B(3) = 1 1 1 1 1 0 1,

B(4) = 1 0 1 1 0 1 0,

s(i)=1, i=1,...,4.

$$\begin{bmatrix}0\\1\end{bmatrix}_{11}\ \begin{bmatrix}0\\1\end{bmatrix}_{12}\ \begin{bmatrix}0\\1\end{bmatrix}_{13}\ \begin{bmatrix}0\\1\end{bmatrix}_{14}\ \begin{bmatrix}0\\1\end{bmatrix}_{15}\ \begin{bmatrix}0\\1\end{bmatrix}_{16}\ \begin{bmatrix}0\\1\end{bmatrix}_{17}$$

$$\begin{bmatrix}0\\0\end{bmatrix}_{21}\ \begin{bmatrix}0\\0\end{bmatrix}_{22}\ \begin{bmatrix}0\\0\end{bmatrix}_{23}\ \begin{bmatrix}0\\0\end{bmatrix}_{24}\ \begin{bmatrix}0\\0\end{bmatrix}_{25}\ \begin{bmatrix}0\\0\end{bmatrix}_{26}\ \begin{bmatrix}0\\0\end{bmatrix}_{27}$$

$$\begin{bmatrix}0\\1\end{bmatrix}_{31}\ \begin{bmatrix}1\\0\end{bmatrix}_{32}\ \begin{bmatrix}1\\0\end{bmatrix}_{33}\ \begin{bmatrix}1\\0\end{bmatrix}_{34}\ \begin{bmatrix}1\\0\end{bmatrix}_{35}\ \begin{bmatrix}1\\0\end{bmatrix}_{36}\ \begin{bmatrix}1\\0\end{bmatrix}_{37}$$

$$\begin{bmatrix}0\\1\end{bmatrix}_{41}\ \begin{bmatrix}0\\1\end{bmatrix}_{42}\ \begin{bmatrix}0\\1\end{bmatrix}_{43}\ \begin{bmatrix}0\\1\end{bmatrix}_{44}\ \begin{bmatrix}0\\1\end{bmatrix}_{45}\ \begin{bmatrix}0\\1\end{bmatrix}_{46}\ \begin{bmatrix}0\\0\end{bmatrix}_{47}.$$

For the result of the search, see the interpretation following the algorithm.

This algorithm has two interesting properties:

1) full parallelism - each cell (i,j) is capable of comparing B(i,j) with the unmasked C(j) to produce local results simultaneously; and

2) full serialism - the final comparison result has to be carried out by scanning these local results serially, most significant bit first.

Ramamoorthy's algorithm is, therefore, bit sequential processing in nature; the time complexity $O(n)$ is the best that it can do.

In the next section, a new threshold search algorithm is presented which utilizes parallelism in a more efficient way, resulting in a much faster algorithm.

3.2 A new threshold search algorithm

In presenting the new algorithm, the associative memory model in Fig. 1 and the related notation will be used. Additional notation will be introduced as required. The memory word organization is based on the word-tree concept which will be explained fully. The definitions and terminology for trees and their traversal to be used follow Knuth [15].

3.2.1 The word-tree concept

Let T(i) be a complete binary tree with n nodes from the n cells at the i-th row of the memory unit.

The mapping of cell (i,j) onto the nodes of T(i) is determined by the following two rules:

Rule 1: endorder numbering the nodes of T(i); and

Rule 2: mapping cell (i,j) onto T(i,j), where T(i,j) is the node of T(i) whose number in endorder is j.

As a result, B(i) is stored in a hardware binary tree T(i) termed the *word-tree*.

Definition 3.1. WT(i,j), the weight of a node T(i,j), is 2^{n-j}.

Definition 3.2. WTR(i,j), the weight of the right subtree TR of a node T(i,j), is the sum of the weights of the nodes in TR.

As an immediate consequence, the following lemma describes an interesting property of the word-tree, and strongly suggests a fast threshold algorithm given in the next section.

Lemma 3.1: For any node T(i,k) in the left subtree TL, and T(i,1) in the right subtree TR of T(i,j), then

1) WT(i,k) > WTR(i,j) + WT(i,j), and

2) WT(i,1) > WT(i,j).

Proof: From Rule 1, the number assigned to any node in TL is less than the numbers assigned to those in TR. Furthermore, any number assigned to the nodes in TR is less than the one assigned to node j. Without loss of generality,

assuming that TL and TR both have s nodes, then (1) the maximum number assigned to a node in TL is d+s, (2) the numbers assigned to nodes in TR are: d+s+1, d+s+2,...,d+2s, (3) the number assigned to node j is d+2s+1 for some integer d, $0 \le d \le n-2s-1$.

Thus,

$$WTR(i,j) + WT(i,d+2s+1) = \sum_{t=1}^{s+1} 2^{n-d-s-t},$$

since,

$$\sum_{t=1}^{s+1} 2^{n-d-s-t} = 2^{n-d-s} - 2^{n-d-2s-1},$$

therefore,

$$\sum_{t=1}^{s+1} 2^{n-d-s-t} < 2^{n-d-s}.$$

The right hand side of the inequality is just the smallest weight of a node in TL; and WT(i,k) is not less than this weight. The second part of the lemma can be proved in a similar manner.

Q.E.D.

3.2.2 The algorithm

Without loss of generality, let n equal 2^k-1, and $d \in \{1,...,k\}$. Define $Q(d)=\{j \mid$ the level of T(i,j) is d, $1 \le j \le n\}$. In the algorithm that follows, $E(i,j-2^{k-d})$ and $G(i,j-2^{k-d})$ represent the output generated by cell $(i,j-2^{k-d})$ which is the left son of cell (i,j) of level d, and E(i,j-1) and G(i,j-1) the output generated by cell (i,j-1) which is the right son of cell (i,j). Both outputs are being taken by cell (i,j) as inputs. E(i,j) and G(i,j)

are the output generated by cell (i,j) to its father. The algorithm is now presented as Algorithm 3.1.

Algorithm 3.1. Threshold search.

```
    Begin
1.    C:=search key word;
2.    M(j):=1, Vj∈{1,...,n};
3.    r(i):=0, Vi∈{1,...,m};
4.    E(i,j):=¬(M(j)·(B(i,j)⊕C(j))), Vj∈Q(k);
5.    G(i,j):=M(j)·B(i,j)·¬C(j),      Vj∈Q(k);
      For d:=k-1 Step -1 Until 1 Do
       Begin
6.      E(i,j):=E(i,j-2^(k-d))·E(i,j-1)·¬(M(j)·(B(i,j)⊕C(j))),
                                                  Vj∈Q(d);
7.      G(i,j):=G(i,j-2^(k-d)) + E(i,j-2^(k-d))·(G(i,j-1) +
                E(i,j-1)·M(j)·B(i,j)·¬C(j)),      Vj∈Q(d);
       End;
    End.
```

At completion of the algorithm, E(i,n), G(i,n) together with s(i) indicate the membership of B(i) as follows:

1). $E(i,n)\cdot s(i)=1 \Rightarrow B(i)\in E$;

2). $G(i,n)\cdot s(i)=1 \Rightarrow B(i)\in G$;

3). $\neg E(i,n)\cdot\neg G(i,n)\cdot s(i)=1 \Rightarrow B(i)\in L$.

Theorem 3.1: Algorithm 3.1 performs a threshold search on m words of (2^k-1) bits each in time $O(k)$.

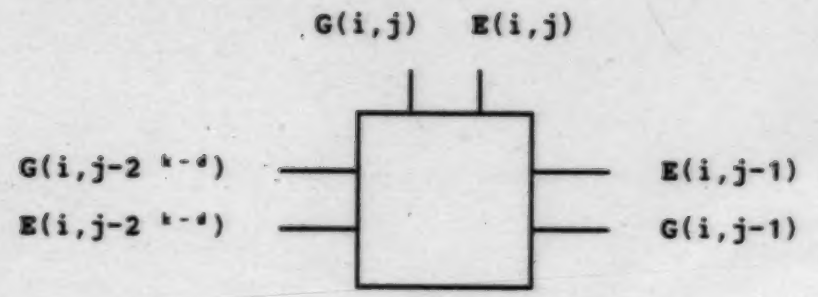
Proof: By induction on k. The basis, k=1,2, is trivial. Now assume the inductive hypothesis is true for k=s. Consider the word tree T(i) with $2^{s+1}-1$ nodes representing B(i) of $(2^{s+1}-1)$ bits, $i \in \{1,\ldots,m\}$. T(i) consists of a root r having left and right subtrees, TL and TR, with 2^s-1 nodes each. Invoking the inductive hypothesis and observing step 6 and 7 of the algorithm (which is based on **Lemma** 3.1), the algorithm works correctly for k=s+1.

One pass of the loop of the algorithm requires constant time (steps 6 and 7) to compute E(i,j) and G(i,j). The loop is repeated k-1 times to give time complexity $O(k)$.

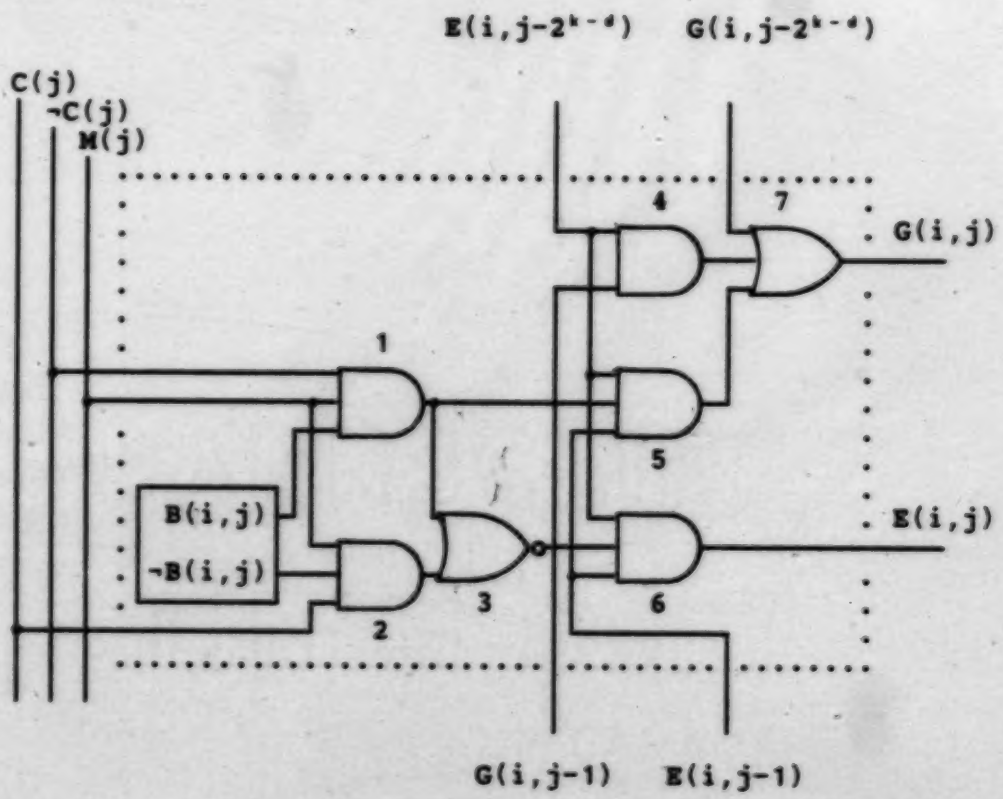
Q.E.D.

Implementation of the algorithm proceeds as follows. The processing logic of the cell is designed to accomplish the computation of step 6 and 7 of the algorithm. Fig. 3(a) shows a cell module used to implement the node of the word-tree. Each cell (i,j) has two incoming lines, $G(i,j-2^{k-d})$ and $E(i,j-2^{k-d})$ from its left son, two incoming lines, G(i,j-1) and E(i,j-1) from its right son, and two outgoing lines, G(i,j) and E(i,j) to its father.

A logical realization of the cell is shown in Fig. 3 (b). Accordingly, steps 6 and 7 of the algorithm can be completed in a period of exactly two gate delays. The amount of comparison logic of the cell is seven gates, with maximum fan-in and fan-out of three.



(a)



(b)

Fig. 3. Searching logic.

Using the cell module as an elementary unit, the memory unit in Fig. 1 can be constructed by m·n identical cells with a uniform interconnection.

It should be pointed out that preorder or postorder numbering of the nodes of T(i), together with an appropriate modification of steps 6 and 7 of the algorithm can serve the fast search purpose equally well.

The following example demonstrates the two-bit outputs G(i,j) and E(i,j) of cell (i,j), for a threshold search of four words of seven bits each. Each word is stored in its corresponding word-tree. In this example, $d \in \{1,2,3\}$, $Q(3)=\{1,2,4,5\}$, $Q(2)=\{3,6\}$, $Q(1)=\{7\}$.

The symbol $\begin{bmatrix} G \\ E \end{bmatrix}_{ij}^{d}$ represents the state of G(i,j) and E(i,j) enabled by cell (i,j) of level d at the end of the (3-d)-th iteration of Algorithm 3.1. For the result of the search, see the interpretation following Algorithm 3.1.

Example: Threshold search operation.

search key word: C = 1 0 1 1 0 1 1,
bit mask register: M = 1 1 1 1 0 1 1,
effective search key word: 1 0 1 1 ϕ 1 1,
words in the memory: B(1) = 1 0 1 1 1 1 1,
B(2) = 0 1 0 0 0 1 0,
B(3) = 1 1 1 1 1 0 1,
B(4) = 1 0 1 1 0 1 0.

$$\begin{bmatrix}0\\1\end{bmatrix}^3_{11}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{12}\ \begin{bmatrix}0\\1\end{bmatrix}^2_{13}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{14}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{15}\ \begin{bmatrix}0\\1\end{bmatrix}^2_{16}\ \begin{bmatrix}0\\1\end{bmatrix}^1_{17}$$

$$\begin{bmatrix}0\\0\end{bmatrix}^3_{21}\ \begin{bmatrix}1\\0\end{bmatrix}^3_{22}\ \begin{bmatrix}0\\0\end{bmatrix}^2_{23}\ \begin{bmatrix}0\\0\end{bmatrix}^3_{24}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{25}\ \begin{bmatrix}0\\0\end{bmatrix}^2_{26}\ \begin{bmatrix}0\\0\end{bmatrix}^1_{27}$$

$$\begin{bmatrix}0\\1\end{bmatrix}^3_{31}\ \begin{bmatrix}1\\0\end{bmatrix}^3_{32}\ \begin{bmatrix}1\\0\end{bmatrix}^2_{33}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{34}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{35}\ \begin{bmatrix}0\\0\end{bmatrix}^2_{36}\ \begin{bmatrix}1\\0\end{bmatrix}^1_{37}$$

$$\begin{bmatrix}0\\1\end{bmatrix}^3_{41}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{42}\ \begin{bmatrix}0\\1\end{bmatrix}^2_{43}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{44}\ \begin{bmatrix}0\\1\end{bmatrix}^3_{45}\ \begin{bmatrix}0\\1\end{bmatrix}^2_{46}\ \begin{bmatrix}0\\0\end{bmatrix}^1_{47}\ .$$

The search is accomplished in 3 computation steps, while Ramamoorthy's algorithm requires 7 steps to do the same job. As seen from this example, a high degree of parallelism has been exploited.

An alternative approach to implementing Algorithm 3.1 makes use of two different kinds of cells for nonterminal nodes and terminal nodes of T(i) respectively. The fact that the terminal nodes have neither left son nor right son allows the removal of gates 4-7 from the original cell. This results in another type of cell with only three gates for the comparison logic which will lie in the position of terminal nodes. This will reduce the hardware complexity. Each cell now requires, on the average, no more than five gates for the processing logic, because the number of terminal nodes is never less than the number of nonterminal nodes.

A natural generalization of the word organization in the form of a complete binary tree is in the form of a complete k-ary tree. In the complete k-ary tree, each node has exactly k sons except for those on the two bottom levels, and each level except the last is completely filled with nonempty nodes. On the last level, some number of rightmost nodes are allowed to be empty.

Mapping a memory word onto the complete k-ary tree can be done in essentially the same way as the complete binary tree.

Definition 3.3. **WT**(i,j,s), the weight of the s-th left subtree of a nonterminal node **T**(i,j), is the sum of the weights of all nodes in the subtree.

An argument similar to the proof of **Lemma** 3.1 can show a useful property of the k-ary word-tree below:

For any node **T**(i,j,s) in the s-th left subtree, $1 \le s \le k$, of a nonterminal node **T**(i,j) of **T**(i) then

$$\mathbf{WT}(i,j,s) > \sum_{l=s+1}^{k} \mathbf{WT}(i,j,l) + \mathbf{WT}(i,j).$$

With the property above a cell module can be designed so that k-ary word-tree can be constructed by n identical cells with uniform interconnections. Each cell has two outgoing lines to its father, and 2k incoming lines from its sons. The function of the cell can be realized by two levels of logic gates. The resulting cell needs (k+5) gates with a maximum fan-in and fan-out of k+1.

It can be easily seen that the search speed increases logarithmically with k, while the hardware complexity of the cell is linear with k. However, the analysis of using two different kinds of cells to reduce the hardware cost is relevant here. In what follows, a formula is derived for the hardware complexity when two types of cells are used to implement the k-ary word-tree.

Suppose the k-ary word-tree has n nodes of which there are i' nonterminal nodes and i terminal nodes, the cell with (k+5) gates is used for the nonterminal node, and the cell with 3 gates for the terminal node. The total number of gates required for the word-tree is $(k+5)\cdot i' + 3i$; hence, the average cost per cell is

$$[(k+5)\cdot i' + 3i]/n. \qquad (1)$$

By the definition of complete k-ary tree, there exists at most one nonterminal node which has fewer than k sons. When this nonterminal node has exactly one son, the relationship between i' and i can be easily established as

$$i = i'\cdot k - (k-1) - (i'-1). \qquad (2)$$

Substituting $n=i'+i$ back into (2), and solving i' in terms of n and k gives

$$i' = (n+k-2)/k, \qquad (3)$$

which implies that $(n-2)$ **mod** $k = 0$ if and only if there exists one nonterminal node which has exactly one son.

However, when $(n-2)$ **mod** $k \neq 0$, an argument to follow will adjust (3) to be of the form

$$i' = \lfloor (k+n-2)/k \rfloor, \qquad (4)$$

where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x.

If $(n-2) \bmod k \neq 0$, then there must exist n', the largest integer belonging to {2,3, ...,n-1} such that $(n'-2) \bmod k = 0$. A k-ary tree of n' nodes will have $(k+n'-2)/k$ nonterminal nodes of which one, say β, has exactly one terminal node, and $n-n'<k$. This will guarantee that developing a k-ary tree of n nodes from that of n' nodes can be done by connecting the (n-n') nodes to β as additional terminal nodes, since the number of sons of β does not exceed k. Therefore, (4) is established.

Substitution of the previous result back into (1) yields the following final form for the average cost per cell: $3 + [(k+2)\cdot(1+\lfloor (n-2)/k \rfloor)]/n$.

In particular, for n=64, k=8, the average cost per cell is 4.25 gates, and the search speed is 6 gate delays.

In conclusion, the use of two types of cells to implement the k-ary word-tree guarantees that search speed increases logarithmically, and hardware complexity decreases slightly as k increases.

3.2.3 Implementation of double-limit searches

Given the above threshold search, all double-limit searches defined in Chapter 2 can be accomplished by executing the proposed algorithm twice in succession. As an example, in order to obtain the search b) of between-limit

searches, the not-less-than search is performed with C=x, followed by the less-than search with C=y on the resulting responders. Other double-limit searches can be obtained in the same way. Consequently, all double-limit searches can also be performed in time $O(\log n)$.

3.3 A new extremum search algorithm

The fast speed of the proposed threshold search guarantees that extremum searches can be achieved by a sequence of threshold searches without introducing undue time penalities. Frie and Goldberg's method for extremum search needs to perform exactly n equality searches [5]. A new extremum search algorithm to be described makes use of a threshold search sequence instead. It will be shown that the number of threshold searches can be expected to be 0.7n.

To begin with, a maximum search algorithm is presented as Algorithm 3.2 to illustrate how the search can be achieved by a sequence of threshold searches.

Algorithm 3.2. Maximum search.

```
      Begin
1       M:=0; j:=1;
2       While j≤n  Do
          Begin
3           C(j):=1; C(j+1):=0; M(j):=1; M(j+1):=1;
4           Threshold-search;
                 m
5           f:= U (G(i,n)·s(i)) (wired-OR);
                i=1
                 m
6           g:= U (E(i,n)·s(i)) (wired-OR);
                i=1
7           If  f=1  Then C(j+1):=1;
8           Else If g=0
              Then
                Begin
9                 C(j):=0;
10                Threshold-search;
                       m
11                f:= U (G(i,n)·s(i)) (wired-OR);
                      i=1
12                If f=1 Then  C(j+1):=1;
                End;
13        j:=j+2;
        End;
    End.
```

The action of this algorithm can be described briefly as follows: step 1 initializes M and j. During the first iteration, after step 3 is executed, M has 1's in only the leftmost two bits starting from the first bit position. The corresponding two bits of C contains 10. Step 4 splits the set of candidate words into the three subsets shown below, according to the search pattern specified by C and M,

11◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆

10◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆

0◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆.

Step 7 checks if the first subset is empty or not. If it is not, the word with the maximum value must have 11 in the leftmost two-bits. Therefore C(j+1) is updated to 1. However, if both the first and second subsets are empty (tested in step 8), then the word with maximum value has to be in the third subset. Accordingly, C(j) is reset to 0 and an extra separation, step 10, is necessary to determine which of the following two subsets contains the word with maximum value,

01◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆

00◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆.

Likewise, if the current first subset is not empty (tested in step 12), C(j+1) is updated to 1. Step 13 increases j by 2, and sets M(j), M(j+1) to 11, C(j), C(j+1) to 10.

In the succeeding iterations, M and C will specify a searching pattern which agrees up to the current (j-1)-th bit with the word having the maximum value and has 10 in the

next two-bit string (the j-th and (j+1)-th bits). The whole process is then repeated, and the loop executed n/2 times. Each time, however, the number of searches is not fixed, but depends on the distribution of 11, 10, 01, and 00 in each two-bit string of the maximum value. Assuming the distribution is even, then the average number of searches for each iteration is: $(0.25)\cdot 1 + (0.25)\cdot 1 + (0.5)\cdot 2 = 1.5$. Upon termination of the algorithm, C(1) through C(n) will hold a copy of the maximum value.

The above algorithm can be generalized so that the register M is filled with k 1's at a time to obtain a family of maximum search algorithms. In order to evaluate these algorithms and select the one of which the number of threshold searches required is minimal, the assumption here is that the bit patterns 00...0, 00...1, ..., 11...1 are evenly distributed in each k-bit string of the maximum value. Let I(k) be the number of searches in the average case, for each loop of the generalized algorithm, then

$$I(k) = 2^{-k}\left[\sum_{i=1}^{k-2} i\cdot 2^{i-1} + (k-1)\cdot(2^{k-2} + 2^{k-1} - 1) + 2k\right]$$

$$= 2^{-k}\left[\sum_{i=0}^{k-2}\sum_{j=i}^{k-2} 2^{j} + (k-1)2^{k-1} + k + 1\right]$$

$$= k - 1.5 + (k+2)\cdot 2^{-k} .$$

Therefore, to implement the maximum search requires a sequence of $I(k)\cdot n/k$ threshold searches. $I(k)/k$ is the ratio of the number of searches required by the proposed algorithm

and that by Frie and Goldberg's algorithm [5]. When k=3, the ratio is 0.7 which proves to be the most efficient choice of all. The number of memory interrogations can thus be reduced approximately by 30% in the average case.

The minimum search can be done, on the same basis, with the same time complexity. Adjancency searches can be achieved by following a threshold search with an extremum search.

3.4 Conclusion

The search algorithms proposed here are quite efficient. The threshold searches and double-limit searches have significantly improved the time complexity over searches of the same type in the literature, from $O(n)$ to $O(\log n)$. An improvement over Frie and Goldberg's extremum search algorithm has been achieved. Implementations of the proposed algorithms have been suggested.

CHAPTER 4

EFFICIENT ORDERED RETRIEVAL ALGORITHMS

The problem of retrieving an ordered list of k words from associative memories of m n-bit words is addressed in this chapter. An efficient new algorithm is proposed along with a cellular logic implementation. The algorithm is of time complexity $O(n+k)$. By contrast, the fast algorithms by Lewin [7] and Ramamoorthy [31] are of time complexities $O(k \cdot \log n)$ and $O(k \cdot n)$, respectively.

4.1 Previous ordered retrieval algorithms

The ordered retrieval of a set of words from an associative memory has long been an interesting research problem [3],[5],[6],[7],[17] and [31]. The first proposal is due to Frie and Goldberg [5]. Independently, Seeber and Lindquist [6] postulated another scheme which utilized a more complicated reporting mechanism to provide the knowledge regarding whether or not there exists exactly one responder to a previous search. The method in [6] requires, in most cases, fewer memory accesses than that of [5] to retrieve, in order, the same number of responders from an associative memories. However, the inherent characteristic of these two algorithms is that the number of memory accesses is dependent on n [7],[11].

The use of an associative memory with a *Bit-Slice Value Indicator* (**BSVI**) was first introduced by Lewin [7], and has achieved the most efficient ordered retrieval algorithm thus far proposed. The **BSVI** is a hardware mechanism which has the ability to determine, for each bit-slice, whether all words of interest store the same bit in this bit-slice or not. In other words, **BSVI** is capable of distinguishing a mixture of 1's and 0's from either only 1's or only 0's stored in that bit-slice. As proven by the inventer, the number of memory accesses necessary to retrieve k responders in order from an associative memory of m n-bit words is exactly 2k-1, independent on n. A simple proof of Lewin's ordered retrieval theorem can be found elsewhere [13].

Some efforts have been made in the past to further reduce the number of memory accesses of Lewin's algorithm by introducing a more powerful **BSVI**. As reported by Miller [12], when the **BSVI** is also able to determine, for each bit-slice, whether or not there exists exactly one distinct bit stored in the bit-slice, the number of memory accesses can be reduced to, at best, k+1 and, at worst, 2k-1. Unfortunately, Miller's method does not necessarily retrieve the k responders in an ascending or descending order; it can only be used for multiple response resolution rather than sorting.

Further utilizing the powerfulness of the **BSVI**, Ramamoorthy has designed the fastest algorithms for extremum searches [31]. Of particular interest, the maximum and

minimum searches can be performed on the memory concurrently. As a consequence, k responders can be isolated in k/2 memory accesses. However, each memory access of Ramamoorthy's algorithm carries out extremum searches requiring $O(n)$ gate delays [31], while that of Lewin's only performs an equality search. The essential dominant part of Lewin's algorithm is the time required to locate the leftmost bit-slice with a mixture of 1's and 0's stored by all selected words each time before the equality search can take place. This locating process is equivalent to the problem of selecting the first logic 1 in a binary vector of length n, and takes time $O(\log n)$ to complete, even if the best algorithms [14],[26] are assumed.

Measured in terms of gate delays, the time complexities of Lewin's algorithm and Ramamoorthy's algorithm are $O(k \cdot \log n)$ and $O(k \cdot n)$, respectively. Clearly, Lewin's algorithm behaves much better.

The disadvantage with these two fast algorithms lies in the fact that the time period between isolating two adjacent responders is not fixed, but depends on the values of the words under consideration.

In what follows, a new ordered retrieval algorithm is presented for an associative memory with BSVI. With this algorithm, the first responder is located in time $O(n)$, and each of the remaining in $O(1)$.

4.2 A new ordered retrieval algorithm

4.2.1 System overview

The simplified associative memory organization shown in Fig. 4 is used to implement a new ordered retrieval algorithm to be presented.

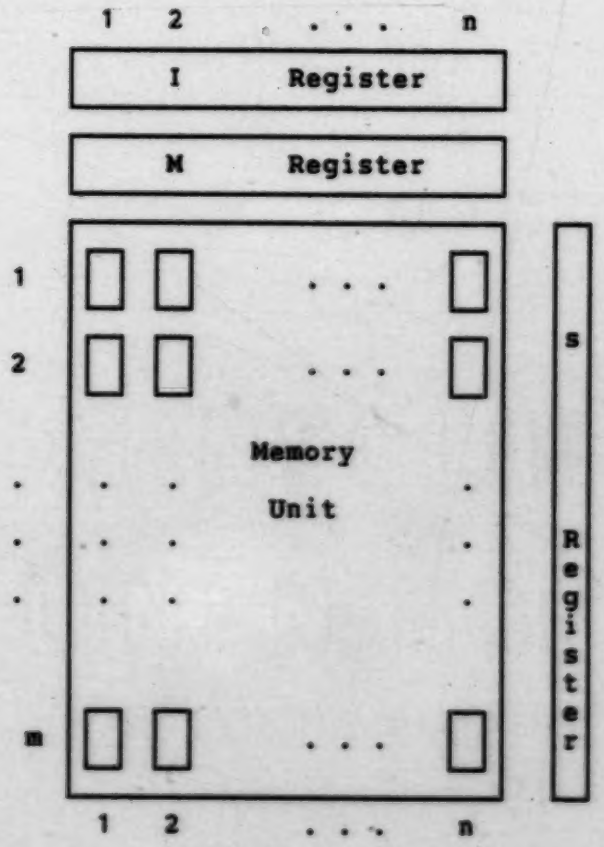


Fig. 4. A simplified associative memory.

As can be seen, the register C has been ignored because no search key word is needed here. The register M is used to mask bit-slices as required. Only those bit-slices for which the corresponding M(j)'s are 1's will be included in the ordered retrieval process. There is another binary matrix A with A(i,j) residing in cell (i,j), which can be visualized as interleaved with the binary matrix B column by column. B(i,1) precedes A(i,1), and A(i,n) follows B(i,n) for all $i \in \{1,\ldots,m\}$.

The s(i) will be denoted throughout this chapter as A(i,0) for descriptive purposes only. Initially, matrix A is reset, and the words to be retrieved in descending order are specified by A(i,0)=1, for some $i \in \{1,\ldots,m\}$.

Informally, it is now instructive to view A(1,j), A(2,j), ... ,A(m,j), $1 \leq j \leq n$, as m paths each of n units long, and the ordered retrieval algorithm as a one-way traffic system, where each logic 1 initially injected into A(i,0), for some $i \in \{1,\ldots,m\}$, matches in path i toward the extreme right end one unit per step. In this regard, each cell (i,j) functions as an intelligent traffic light for path i at a position j units away from A(i,0) and is capable of permiting or inhibiting the passage of the matching logic 1 through it, based on the information carried by bit-slice j. The system is designed in such a way that the order in which the logic 1's reach the very right end is the desired reading sequence of their associated words. Moreover, after n-2 steps, the logic 1's start reaching the very right end

in that order at the rate of one arrival every two steps.

In presenting the algorithm more precisely, the following terminology is useful and introduced in the context of the simplified associative memory. Let $S'=\{B(i)|A(i,j)=1,\ 1\leq i\leq m \text{ and } 0\leq j\leq n\}$ be a set of words to be read out of the memory in descending order in the course of ordered retrieval. For any word $B(i)\epsilon S'$, define a function $P(B(i))$ to be an integer $\epsilon\{0,\ldots,n\}$ such that $A(i,P(B(i)))=1$.

Definition 4.1. The effective j-th bit-slice is made up of the j-th bit of every word B(i) for which $A(i,j-1)=1$, $i\epsilon\{1,\ldots m\}$.

Definition 4.2. The j-th bit-slice is said to be busy if there exists i, $i\epsilon\{1,\ldots,m\}$, such that $A(i,j)=1$.

Definition 4.3. Two words B(1), B(k) are said to be adjacent, if $P(B(1)) > P(B(s))$, and there does not exist $B(k)\epsilon S'$ such that $P(B(1)) < P(B(k)) < P(B(s))$.

Example: Consider the memory unit below, where the symbol $\begin{bmatrix} b \\ a \end{bmatrix}_{ij}$ represents cell (i,j) with a and b for B(i,j) and A(i,j), respectively:

$A(1,0)=0$ $\quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{11} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{12} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{13}$

$$A(2,0)=0 \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}_{21} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{22} \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{23}$$

$$A(3,0)=0 \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}_{31} \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{32} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{33}$$

$$A(4,0)=1 \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{41} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{42} \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{43}$$

$$A(5,0)=1 \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{51} \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{52} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{53}$$

$$A(6,0)=0 \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{61} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{62} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix}_{63},$$

a) The effective 1st, 2nd, and 3rd bit-slice are now

$$\begin{bmatrix} \phi \\ \phi \\ \phi \\ 0 \\ 0 \\ \phi \end{bmatrix}, \quad \begin{bmatrix} \phi \\ 1 \\ 0 \\ \phi \\ \phi \\ \phi \end{bmatrix} \text{ and } \quad \begin{bmatrix} \phi \\ \phi \\ \phi \\ \phi \\ \phi \\ \phi \end{bmatrix}$$

repectively.

b) Only the 1st bit-slice, among the three, is busy.

c) Since P(B(2))=P(B(3))=1, and P(B(4))=P(B(5))=0, both words B(2) and B(3) are adjacent to B(4) and B(5).

The BSVI required here has the following characteristics. First, it is capable of detecting whether or not there exists at least one bit in the effective j-th bit-slice whose value is 1. Second, it is able to determine whether or not the j-th bit-slice is busy.

Using the "wired-or" technique, such a detection can be done in constant time [31].

The j-th component of the bit-slice value indicator and the associated bit-slice are schematically illustrated in Fig. 5, where $h(i,j)=\mathbf{A}(i,j-1)\cdot\mathbf{B}(i,j)$, and $v(j)=\mathbf{A}(i,j)$.

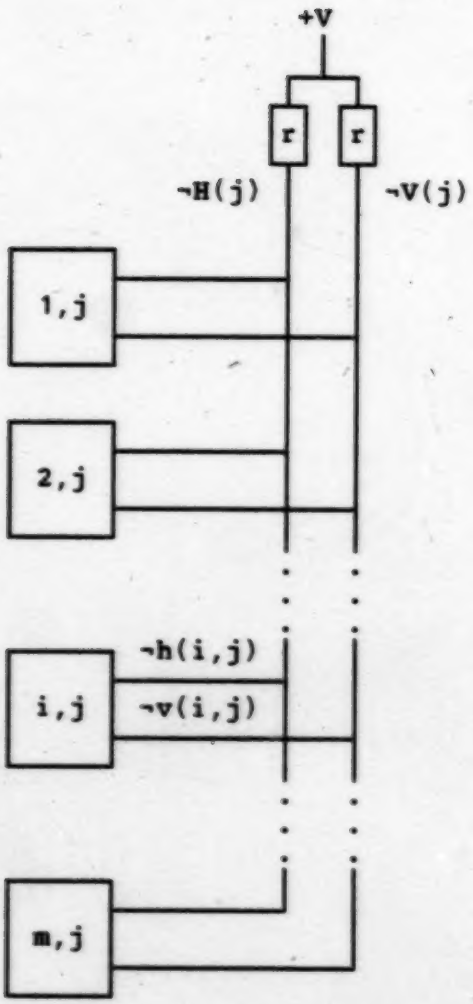


Fig. 5. The j-th component of the BSVI.

It can be readily seen from Fig. 5, that the j-th bit-slice is busy if $\neg V(j)=0$, and that at least one bit of the effective j-th bit-slice is 1 if $\neg H(j)=0$.

In refering to the previous example, the following equations hold: $\neg H(1)=1$, $\neg V(1)=0$; $\neg H(2)=0$, $\neg V(2)=1$; and $\neg H(3)=1$, $\neg V(3)=1$.

4.2.2 The algorithm

The heart of the ordered retrieval algorithm is a parallel operation called partition which is presented below as Algorithm 4.1.

Algorithm 4.1. Partition.

```
        Begin
1       V(j):= U_{i=1}^{m} A(i,j) (wired-OR);
2       H(j):= U_{i=1}^{m} A(i,j-1)·B(i,j) (wired-OR);
3       p(i,j):= A(i,j-1)·(B(i,j)+¬H(j)+¬M(j));
4       If V(j)=0 and p(i,j)=1 Then
          Begin
5           A(i,j):=1;
6           A(i,j-1):=0;
          End;
        End.
```

The essence of Algorithm 4.1 is to simultaneously partition each class of words B(i) indicated by A(i,j-1)=1,

for $\forall i \in \{1,\ldots,m\}$ and $\forall j \in \{1,\ldots,n\}$, into two subclasses based on the information carried by the effective j-th bit-slice. And if the j-th bit-slice is not busy at that moment, then the words B(1) in the bigger magnitude subclass can be separated from the words B(s) in the smaller magnitude subclass by means of setting A(1,j), and resetting A(1,j-1). As a result, the words B(1) and B(s) are now registered by A(1,j+1)=1 and A(s,j)=1 into two subclasses respectively, which were in the same class before the effective j-th bit-slice was examined.

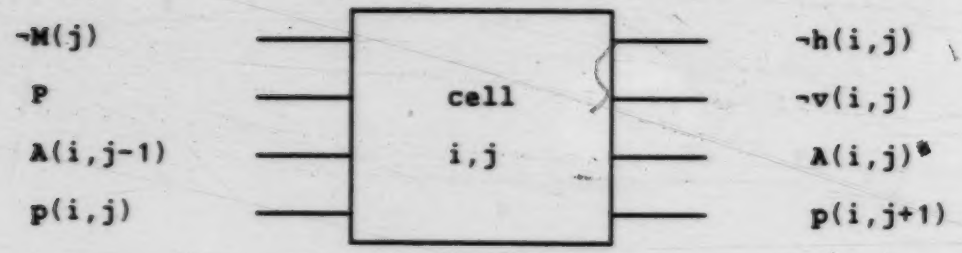
In refering to the partition algorithm shown, steps 1 and 2 compute V(j) and H(j) for all $j \in \{1,\ldots,n\}$ simultaneously. V(j)=1 implies that the j-th bit-slice is busy. H(j)=1 indicates that there exists at least one bit in the effective j-th bit-slice whose value is 1. Step 3 assigns each p(i,j) a logic value. It may be noted from the statement in line 3 that p(i,j) will have the value 0 if either A(i,j-1)=0 or H(j)=1, B(i,j)=0 and the j-th effective bit-slice is unmasked. It is apparent that the word B(i) with A(i,j-1)=1 and p(i,j)=1 is greater than the word B(k) with A(k,j-1) and p(k,j)=0. Moreover, if the j-th bit-slice is free and B(i) belongs to the bigger magnitude subclass, checked by step 4, then the states of A(i,j-1) and A(i,j) are interchanged by steps 5 and 6. This interchange effectively separates the words in the bigger magnitude subclass from those in the smaller magnitude subclass. It is also possible that there does not exist any p(i,j)=0

implying that the smaller magnitude subclass is empty. In this case after steps 5 and 6 are executed, the (j-1)-th bit-slice will be released.

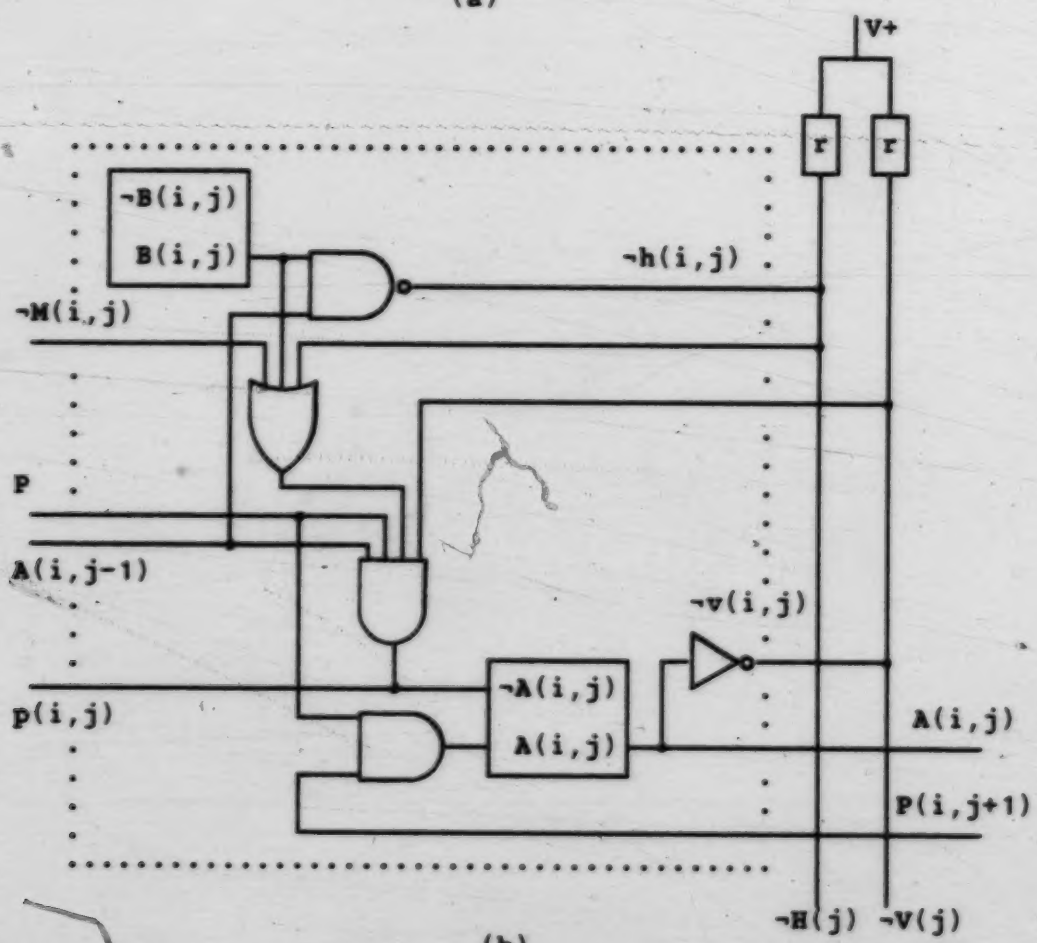
When the j-th bit-slice is busy, however, no separation can be expected to be done due to no room currently available to register the words in the bigger magnitude subclass. Therefore all words B(i) indicated by A(i,j-1)=1 have to remain in the original class as if the effective j-th bit-slice were not examined.

An implementation of the partition algorithm in each cell is shown in Fig. 6. Each cell (i,j) communicates with its left neighbor by means of A(i,j-1) and p(i,j), and with its right neighbor by means of A(i,j) and p(i,j+1). Lines ¬h(i,j) and ¬v(i,j) are coupled to the j-th component of the BSVI as depicted in Fig. 5. Each cell will receive a global timing clock, denoted as P here, to set/reset A(i,j). The partition operation can be accomplished in a constant period of time, 5 gate delays.

The ordered retrieval algorithm is now presented as Algorithm 4.2.



(a)



(b)

Fig. 6. Partition logic.

Algorithm 4.2. Ordered Retrieval.

```
        Begin
1         A(i,j):=0, ∀i∈{1,...,m}, ∀j∈{1,...,n};
2         While V(n)=0 Do Partition;
3         Read out B(i) indicated by A(i,n)=1;
4         Partition;
5         A(i,n):=0;
6         While V(n-1)=1 Do
            Begin
7             Partition;
8             Read out B(i) indicated by A(i,n)=1;
9             Partition;
10            A(i,n):=0;
            End;
        End.
```

Algorithm 4.2 is achieved essentially by performing a sequence of partition operations to isolate all words in S' by their contents, and read them out one at time without ambiguity. It is assumed that all words in the memory are different in order to identify them uniquely.

Initially, A(i,j)=0, for all $i\in\{1,\ldots,m\}$, and all $j\in\{1,\ldots,n\}$, and A(i,0)=1, for some $i\in\{1,\ldots,m\}$. A(i,0)=1 indicates word B(i) in S'. The algorithm terminates when all responders have been read out of the memory. In other words, when the set S' becomes empty.

4.2.3 An analysis of the algorithm

In order to analyze the time complexity, and verify the validity of Algorithm 4.2, three lemmas are first established concerning the effectiveness of a sequence of partition operations on the words in S'.

Lemma 4.1: The relation P(B(1)) > P(B(s)) is not affected by any partition to be executed.

Proof: Consider the following two cases.

CASE 1. Suppose P(B(1)) - P(B(s)) = 1.

P(B(s)) can be increased by 1 iff steps 5 and 6 are executed. However, steps 5 and 6 cannot be executed in this case, because V(P(B(1))) equals 1.

CASE 2. Suppose P(B(1)) - P(B(s)) > 1.

P(B(s)) can be increased at most by 1 after steps 5 and 6 are executed. However, this increment by no means changes the realtion P(B(1)) > P(B(s)).

Q.E.D

Lemma 4.2: B(1) > B(s) if P(B(1)) > P(B(s)).

Proof: There are only two ways in which P(B(1)) > P(B(s)) can occur after a partition P is executed.

CASE 1. Before executing P, P(B(1)) = P(B(s)) which implies that B(1,j) = B(s,j) for j=1,2,...,P(B(1)). After P is executed, condition P(B(1)) > P(B(s)), implies B(1,P(B(1))) = 1 and B(s,P(B(1))) = 0 for the increased P(B(1)). Therefore, B(1) > B(s) if P(B(1)) > P(B(s))

follows.

CASE 2. Before executing *P*, P(B(1)) > P(B(s)), which recursively implies that separating B(1) from B(s) was due to an earlier partition *P'*. In other words, there exists an h such tnat B(1,j) = B(s,j), for j=1,2,...,h-1, and B(1,h)=1, B(s,h)=0; the separation was done by *P'* based on the above information carried by the effective h-th bit-slice. By **Lemma** 4.1, B(1) > B(s) if P(B(1)) > P(B(s)) follows.

Q.E.D.

Lemma 4.3: P(B(1)) - P(B(s)) ≤ 2 if B(1) and B(s) are adjacent.

Proof: The lemma can be proven by showing that B(1) and B(s) are not adjacent if P(B(1)) - P(B(s)) = k (k>2), by induction.

The basis, k=3. There must exist a partition *P* such that P(B(1)) - P(B(s)) = 2 before *P* takes place. At completion of *P*, P(B(1)) - P(B(s)) = 3 implies that P(B(1)) is increased by one but not P(B(s)). However, P(B(s)) remains unchanged only when either V(P(B(s))+1) = 1 or p(s,P(B(s))+1) = 0 during the execution of *P*. The first condition implies that B(1) and B(s) cannot be adjacent due to the existance of B(k) such that P(B(1))>P(B(k))>P(B(s)), and that they can never become adjacent, after the execution of *P*, by **Lemma** 4.1. The second condition implies that there exists at least one word B(k') such that P(B(k')) = P(B(s))

and p(k',P(B(s))+1) = 1. Consequently, B(1) and B(s) will be no longer adjacent, because P(B(1)) > P(B(k')) > P(B(s)) must occur after P is executed.

Now assume that the inductive hypothesis is true for all values of k≥3. Consider P(B(1)) - P(B(s)) = k+1. Again, there must be a partition P' such that P(B(1)) - P(B(s)) = k before it takes place. By the inductive hypothesis, B(1) and B(s) cannot be adjacent. By **Lemma** 4.1, they can never become adjacent, even after P' is executed.

Q.E.D.

In what follows, it is assumed that the read operation transmitting the contents of a single word B(i) indicated by A(i,n)=1 into the register I can be done in constant time [28]. Also it is assumed that this read is executed in parallel with the partition following it in Algorithm 4.2 even though it has been written in a serial manner. The time complexity of Algorithm 4.2 will be measured in terms of the number of partition operations.

Theorem 4.1: The number of partitions required by Algorithm 4.2 to retrieve k responders in descending order from the associative memory of m n-bit words is n+2k-1.

Proof: After step 1 is executed, the words in S' are indicated by A(i,0)=1 only. None of the bit-slices 1 through n is busy before a partition is invoked.

The number of iterations of the first **While** loop in line 2 is n. In other words, to isolate the first responder from the rest of them requires n partitions. The fact that bit-slices through j to n are all free before the j-th iteration takes place, where $1 \leq j \leq n$, gurantees that there must exist one word, say B(i), being registered by A(i,j) with j increasing each time a partition is executed. At the completion of the n partitions, exactly one A(i,n) associated with this word is set, which indicates that the word can be read out immediately without ambiguity.

After step 4 is executed, by **Lemma** 4.3, A(i',n-1), for some $i' \in \{1,\ldots,m\}$, will be set to register words B(i') which are adjacent to the word indicated by A(i,n)=1. Step 5 resets A(i,n) making the n-th bit-slice no longer busy.

The number of iterations of the next **While** loop is k-1. During the first iteration, after step 7 is performed, a new B(i) will be registered by A(i,n)=1, and be read out subsequently. Step 9 guarantees that the words adjacent to B(i), say B(i'), will be registered by A(i',n-1)=1, after it takes effect. Step 10 clears up A(i,n) discarding this B(i) from further consideration and, releasing the n-th bit-slice. In the succeeding iteration of the loop, the words B(i') are registered by A(i',n-1)=1, the n-th bit-slice is free, and all words in S' satisfy the property specified by **Lemma** 4.3. The loop will be repeated k-1 times to exhaust the remaining k-1 responders.

By **Lemma** 4.2, the k words are retrieved in descending order.

Taking into account the total cost above gives that n+2k-1 partitions are necessary.

The correctness of the algorithm follows by induction on k.

Q.E.D.

4.3 Conclusion

An efficient ordered retrieval algorithm has been presented which retrieves k responders in descending order from an associative memory of m n-bit words in $O(n+k)$ steps. It possesses the salient feature that after the first responder is obtained, each of the remaining can be found in constant time.

CHAPTER 5

FAST MULTIPLE RESPONSE RESOLUTION ALGORITHMS

The problem of multiple response resolution is discussed in this chapter. Alternative techniques for solving this problem are briefly reviewed with the emphasis on Anderson's method [25]. A new method is outlined based on the fast ordered retrieval algorithm proposed in Chapter 4. A comparison is then made between Anderson's method and the proposed one.

5.1 Previous results

Consider the associative memory organization shown in Fig. 1. A search of the associative memory, in general, separates the stored m words into two subsets of which one contains all the words satisfying the search criteria. The words in this subset are also sometimes called responders. Let S, defined in Chapter 2, be this subset, i.e., s(i)=1 indicates that B(i) is a responder. Multiple response resolution occurs when the associative memory must read out these responders one at a time for the purpose of output. To select a responder to read, the associative memory must be able to choose from the m-bit vector, register s, an element s(i) which is logic 1. In other words, it must be able to generate from s an m-bit vector R of which only one element is 1 corresponding to the selected responder. Identified by

the vector R, the selected word can be read out without ambiguity. How fast the vector R can be generated will dominate the output speed of associative memories because the vector R has to be generated as many times as the number of responders.

Several fast methods for generating the vector R have been proposed in the past, and they can be grouped into two classes according to whether or not further associative searches on the responders are required. The first class, including methods due to Frie and Goldberg [5], Seeber and Lindquist [6], Lewin [7] and Ramamoorthy [31], uses further associative searches to retrieve the responders in an ordered sequence according to their contents. The responders are supposed to be distinct so that no futher multiple response conflict can occur.

The second class comprising those of Weinstein [10], Koo [20], Foster [14], Anderson [26] and Landis [29] utilizes priority circuitry to retrieve them in an ordered sequence according to their physical location in the memory. It is desired in this case to select sequentially all logic 1's in register s one at a time. The simplest way of doing this, as mentioned by Weistein[10], is to scan serially all bits of the register s. If s(i) is not the "first" logic 1, a logic 0 will be generated for R(i), inhibiting read out of B(i). The scheme requires $O(m)$ gate delays to generate the vector R which will defeat the main advantage of a parallel search of associative memories when m becomes large.

An alternative method with time complexity $O(\log m)$ was first proposed by Weistein using tree structured logic to generate the vector **R** [10]. Weistein's resolver operates in two phases: in the first, pulses propagate up the tree, setting storage elements on the way, and in the second, these storage elements steer a downwards propagating pulse to the "first" of the responders.

Later, a faster multiple response resolver embodying lookahead logic was presented by Foster[14]. Foster's resolver uses much less hardware than Weistein's and generates the vector **R** approximately two times as fast. Foster's scheme was then generalized by Landis [29] to achieve a resolving speed about 20 percent faster. The most efficient approach using tree structured logic for generating the vector **R** is due to Anderson [26]. Anderson's resolver can speed up Foster's scheme at least by a factor of two, and becomes the fastest among the second class thus far developed.

It has been widely believed that Foster's scheme is much faster in absolute speed than the ordered retrieval methods [29],[31]. The fast ordered retrieval algorithm proposed in Chapter 4, however, is very promising for resolving multiple responses. Such a resolver is suggested in Section 5.3. It is then only neccessary to compare it with Anderson's resolver. The following section is therefore devoted to describing Anderson's method.

5.2 Anderson's Multiple Response Resolver

Fig. 7 shows Anderson's tree structured resolver. The tree is constructed with identical logic blocks of 4 input pairs and 1 output pair. The function of each block is to perform the resolution operation when enabled to do so by a parent block above it, and to inform the parent of any 1 bits (activity) on its inputs when its parent enables the resolution, An indication vector is passed to its descendent blocks (or to the vector R), enabling their resolution operation.

Each interconnecting pair is comprised of an activity signal propagating up the tree to the root block and an indication signal propagating down. The pairs at the terminal nodes of the tree are the register s and the vector R, while the pair at the root are the system level activity and enable signals.

Fig. 8 shows the circuit Anderson calls the "P-generator" block. The A00 - A11 lines represent inputs of activity from four lower locations in the tree; four bits of s or four descendent blocks depending on the level. If any of them are 1, then the A-out line going up the tree is energized. The P00 - P11 lines represent outputs of the indication vector to four lower locations in the tree.

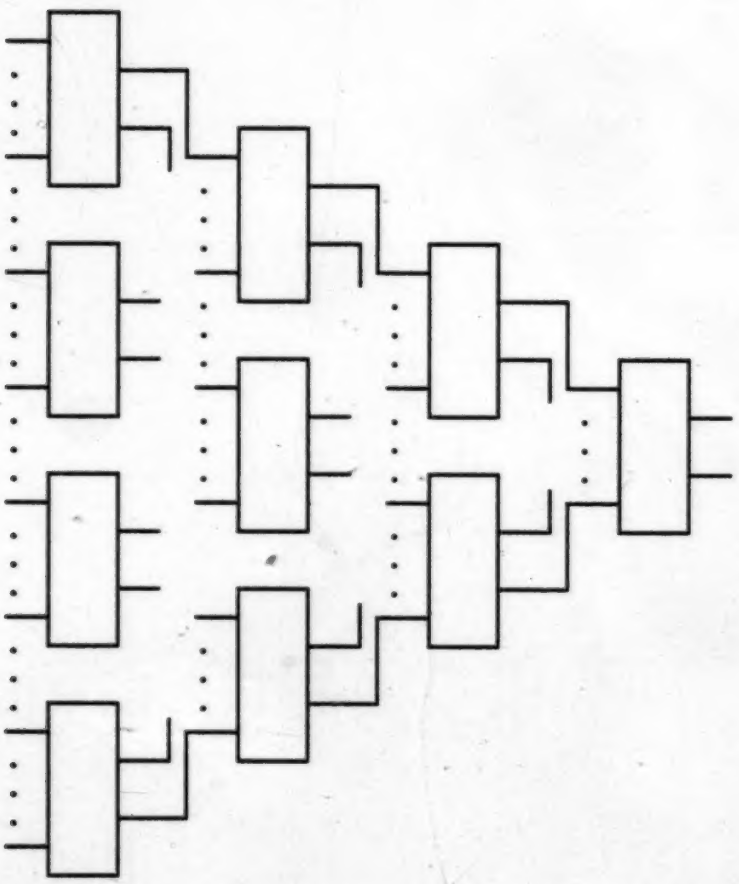


Fig. 7. Anderson's resolver Structure.

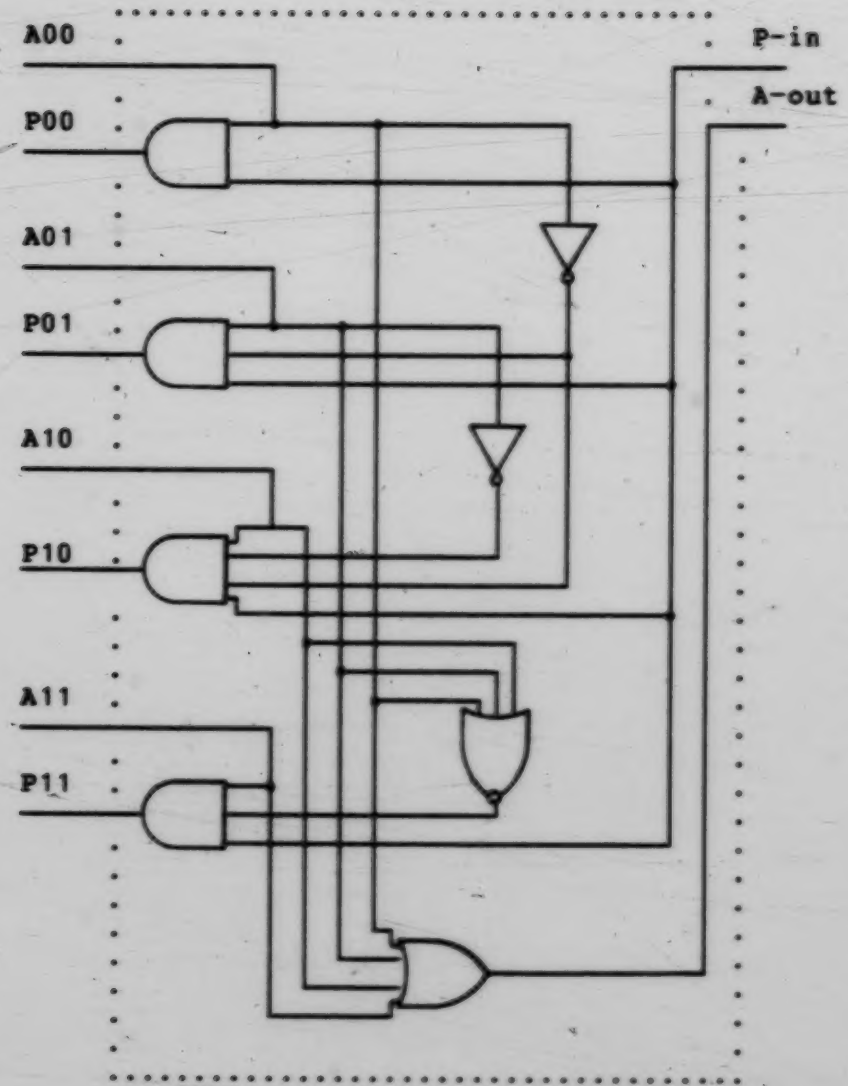


Fig. 8. P-generator block.

The time for generating R vector is calculated as follows. For a memory of m words, Anderson's tree will have (log m)/2 levels. The activity signal must propagate upwards in the tree through (log m)/2 levels for the root block to generate the enable signal. This signal has to propagate downwards in the tree through (log m)/2 levels to generate the R vector. This gives (log m) unit delays in total. The time complexity of Anderson's method is, therefore, $O(k \cdot \log m)$. It should be noted that the major advantages of Anderson's resolver are its very fast speed and its cellular structure.

5.3 A new multiple response resolver

The ordered retrieval algorithm introduced in Chapter 4 is directly applicable to solving the problem of multiple response resolution. A tag field is included for each word, where each tag is a distinguishable number of length log m. When multiple responses occur, each tag serves as a number for the ordered retrieval algorithm. The tags involved in the ordered retrieval are those corresponding to the response words. Only the bit-slices composing the tag field are used in the process. Consequently, this scheme requires $O(\log m + k)$ steps to resolve k responders.

5.4 Comparison with Anderson's resolver

The distinguishing feature of the multiple response resolver outlined in the previous section is that the vector R can be generated in constant time for each of the responders except the first one. This implies that it is potentially attractive for large associative memories. It would be of great value to evaluate the proposed method in terms of Anderson's approach which has been widely regarded as the fastest one.

In order to precisely compare these two schemes, it is assumed that (1) reading of a selected word into register I can be done in constant time [28] and (2) I is made up of *D* type flip flops.

For a memory of size m, Anderson's resolver requires log m gate delays to generate the vector R. Once the vector R has been generated, one word, say B(i), indicated by R(i)=1 can be read into register I. s(i) is then reset, discarding B(i) from further consideration. The process is repeated until all responders have be exhausted.

Reading a selected word B(i) and generating another vector R for the next word can be overlaped in time. They cannot, however, start at the same time, for the following reasons. First, s(i) has to be reset to guarantee that the vector R to be generated would not locate the same B(i) again. Second, resetting s(i) has to be done in such a way that the currently used vector R cannot be disturbed before register I can receive B(i) properly. It is reasonable to

assume that generating the next vector R can be started 4 gate delays later than the read operation, where 4 is due to the *set up* time and *hold* time for the *D* flip flop as well as the reset time for s(i). Further suppose a memory read operation takes time less than (log m +4) gate delays, therefore the total time required by Anderson's method to resolve k responders is k·(log m +4) gate delays. The total time needed by the proposed resolver to do the same task is 5(log m +2k-1). Consequently, when k is greater than 5(log m -1)/(log m -6) the proposed resolver takes less time than Anderson's scheme does. Table I gives a list of values for log m, k' and the ratio k'/m, where k'=5(log m -1)/(log m -6). As the table implies, the proposed method becomes more efficient than its competitor when the memory size is large. For example, when m= 4096, the new resolver always takes less time than its competitor as long as k is greater than 9. For the purpose of output, however, to read a list of more than 9 out of the 4096 words would be the most likely case.

Table I

log m	k'	k'/m
7	30	0.234375
8	17	0.066406
9	13	0.025391
10	11	0.010742
11	10	0.004883
12	9	0.002197
13	8	0.000979
14	8	0.000488
15	7	0.000213
16	7	0.000106
17	7	0.000053
18	7	0.000027
19	7	0.000013
20	6	0.000006

CHAPTER 6
CONCLUSION

Three major types of algorithms for parallel associative memories have been investigated in this thesis. Effecient algorithms for searching, ordered retrieval as well as multiple response resolution have been proposed. Each of the new algorithms has been evaluated in terms of the best algorithm of the same type in the literature. A comparison of the results has shown that the proposed algorithms are superior in most cases.

BIBLIOGRAPHY

1. A. E. Slade and H. O. McMahon, "A cryotron catalog memory system," *Proc. Easten JCC*, vol. 10, pp. 115-120, Dec. 1956.

2. W. Shooman, "Parallel processing with vertical data," *Proc. Easten JCC.*, 1960.

3. R. R. Seeber, "Associative self-sorting memory," *Proc. Eastern JCC*, vol. 18, pp. 179-187, Dec, 1960.

4. E. J. Gauss, "Locating the largest word in a file using a modified memory," *JACM*, vol. 8, pp. 418-425, July 1961.

5. E. H. Frei and J. Goldberg, "A method for resolving multiple responses in a parallel search file," *IRE Trans. Comput.*, vol. EC-10, pp. 718-722, Dec. 1961.

6. R. R. Seeber and A. B. Lindquist, "Associative memory with ordered retrieval," *IBM Journal*, vol. 6, pp. 126-136, Jan. 1962.

7. M. H. Lewin, "Retrieval of ordered list from a content-addressed memory," *RCA Rev.*, vol. 23, pp. 215-229, June 1962.

8. A. D. Falkoff, "Algorithms for parallel-search memories," *JACM* , vol. 9, pp. 488-511, Oct. 1962.

9. G. Estrin and R. H. Fuller, "Algorithms for content-addressable memories," *IEEE pacific computer conf.*, pp. 118-130, 1963.

10. H. Weinstein, "Proposals for ordered sequential detection of simultaneous multiple responses," *IEEE Trans. Comput.*, vol. EC-12, pp. 564-567, Oct. 1963.

11. L. R. Johnson and M. H. McAndrew. "On ordered retrieval from an associative memory," *IBM Journal*, vol. 8, pp.

189-193, April, 1964.

12. H. S. Miller, "Resolving multiple responses in an associative memory," *IEEE Trans. Comput.*, vol. EC-13, pp. 614-616, Oct. 1964.

13. A. Wolinsky, "A simple proof of Lewin's ordered retrieval theorem for associative memory," *CACM*, vol. 11, pp. 448-490, July 1968.

14. C. C. Foster, "Determination of priority in associative memories," *IEEE Trans. Comput.*, vol. EC-17, pp. 788-789, Aug. 1968.

15. D. E. Knuth, *The art of computer programming*, vol. 1, *Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.

16. D. E. Knuth, *The art of computer programming*, vol. 2, *Sorting and Searching*, Addison-Wesley, Reading, Mass., 1969.

17. V. H. Kautz, "Cellular logic-in-memory arrays," *IEEE Trans. Comput.*, vol. C-18, pp. 719-727, Aug. 1969.

18. A. Wolinsky, "Unified interval classfication and unified 3-classfication for associative memory," *IEEE Trans. Comput.*, vol. C-18, pp. 899-911, Oct. 1969.

19. T. Y. Feng and C. C. Lee, "Associative memory manipulations," *Proc. Hawaii International Conf. on System Sciences*, pp. 833-837, Jan. 1970.

20. J. T. Koo, "Integrated circuit content addressable memories," *IEEE Trans. Solid State Circuits*, pp. 208-215, Oct. 1970.

21. J. Minker, "An overview of associative or content-addressable memory systems and KWIC index to the literature," *Computing Reviews*, vol. 12, pp. 453-504, Oct. 1971.

22. C. C. Foster and F. Stockton, "Counting responders in an

associative memory," *IEEE Trans. Comput.*, vol. C-20, pp. 1580-1584, Dec. 1971.

23. B. Parhami, "Associative memories and processors: An overview and selected bibliography," *Proc. IEEE*, vol. 61, pp. 722-730, June 1973.

24. D. W. Digby, "A search memory for many-to-many comparisons," *IEEE Trans. Comput.*, vol. C-22, pp. 768-772, Aug. 1973.

25. C. DeFiore and P. B. Berra, "A quantitative analysis of the utilization of associative memories in data base management," *IEEE Trans. Comput.*, Vol. C-23, pp. 121-123, Feb, 1974.

26. G. A. Anderson, "Multiple match resolvers: a new design method," *IEEE Trans. Comput.*, vol. C-23, pp. 1317-1320, Dec. 1974.

27. S. S. Yau and H. S. Fung, "Associative processor architecture- A survey," in *Proc. 1975 Sagamore Computer Conf. on Parallel Processing, IEEE Computer Society*, pp. 1-14, Aug. 1975.

28. C. C. Foster, *Content addressable Parallel Processors*, New York: Van Nostrand Reinhold, 1976.

29. D. Landis, "Multiple-response resolution in associative systems," *IEEE Trans. Comput.*, vol. C-26, pp. 230-235, Mar. 1977.

30. I. N. Chen, "Performing summation and product in an associative processor," *Proc. of the 1977 International Conf. on Parallel Processing*, pp. 155-156, Aug. 1977.

31. C. V. Ramamoorthy, J. L. Turner, and B. W. Wah, "A design of a fast cellular associative memory for ordered retrieval," *IEEE Trans. Comput.*, vol. C-27, pp. 800-815, Sept. 1978.

32. W. A. Davis and D. L. Lee, "Fast search algorithms for associative memories," to appear in *IEEE Trans. Comput.*, 1984.

33. W. A. Davis and D. L. Lee, "An associative memory scheme," to appear in *Proc. the 1st International Conf. on Computers and Applications*, 1984.